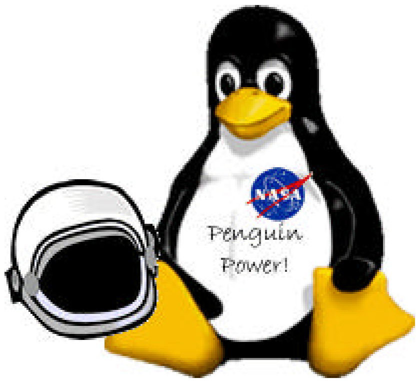
FlightLinux Project Final Report

Patrick H. Stakem

QSS Group, Inc.

June 20, 2002



Revision History

6/19/2002	Initial Release

Introduction and Background

This is the Final Report on the two-year FlightLinux implementation effort. This work was conducted under task NAS5-99124-297, with funding by the National Aeronautics and Space Administration (NASA) Advanced Information Systems Technology (AIST) Program, NRA-99-OES-08. The work is conducted by personnel of QSS Group, Inc. in partnership with Goddard Space Flight Center (GSFC) code 586 (Science Data Systems), code 582 (Flight Software) and code 588 (Advanced Architecture & Automation). Work continued on this task under contract NAS5-99124-564, and concludes on June 30, 2002.

The FlightLinux project had the stated goal of providing an on-orbit flight demonstration of the Linux software, resulting in a Technology Readiness Level (TRL) of 7. The FlightLinux proof-of-concept demonstration was done in conjunction with the on-orbit UoSat-12 mission, from Surrey Space Technology, Ltd. (SSTL). The proposed platform was the WideField Infrared Explorer (WIRE) spacecraft. After extensive discussions with the WIRE Project and software developers, this approach was abandoned due to the complexity involved. We then explored a spectrum of options, including the Surrey Nanosat Applications Platform (SNAP) mission, and potential GSFC near-term missions. We needed to identify a mission that was close enough to launch to allow us to meet our timetable for on-orbit testing, but also one that was amenable to hosting our code. The only option we found was the SSTL UoSat-12 mission, currently in orbit. Because of the use of multiple flight computers, we could load and test our code on-orbit without unduly impacting or interfering with the operations of the spacecraft.

The Orbiting Missions as Nodes on the Internet (OMNI) project of Code 588 at GSFC obtained a breadboard of the Surrey onboard computer (OBC) that was used for testing. In addition, telecommunications facilities at Building 23 at GSFC allowed direct communication with the UoSat-12 spacecraft.

Because almost all of the efforts in developing onboard computer hardware for spacecraft involves adapting existing commercial designs, the logical next step is to adapt commercial off-the-shelf (COTS) software, such as the Linux operating system. Given Linux, many avenues and opportunities become available. Web serving and file transfers become standard features. Onboard local area network (LAN) and an onboard file system become "givens." The Java language is trivial to implement. Commonality with ground environments allows rapid migration of algorithms from ground-based to the flight system, and tapping into the worldwide expertise of Linux developments provides a large pool of talent. Full source for the operating system and drivers is available on day one of the project.

Since we posted our goals of keeping the FlightLinux open source, within the meaning of the gnu license (www.gnu.org/license), we have had numerous offers of collaboration on the project. These include representatives of worldwide aerospace companies, educational institutions, and individuals. The interest in the FlightLinux Project is growing, due to increasing exposure of the website. We are aware of more than a dozen flight projects using Linux. These and other Outreach efforts are documented later in this report.

FlightLinux enables the Office of Earth Science (OES) Sensor Web concept by allowing application-level connectivity among constellations of Earth orbiting satellites, and commonality of ground and space-based environments for ease of application migration.

What is FlightLinux? It is a specific build of the Linux operating system, with unneeded features removed, and spacecraft-specific device drives included. It is "tuned" for the spacecraft environment, and has a minimized footprint. It can be based on any of the standard Linux distributions (RedHat, Mandrake, etc).

An initial build of the FlightLinux software has been running since March 2001. The UoSat-12 breadboard has been available at the OMNI Lab since April 2001. The breadboard has an associated Windows NT machine to load software via the Controller Area Network (CAN) bus or the asynchronous port and to provide debugging visibility. From our facility at QSS, we can access the breadboard facility via the cots "PC-Anywhere" software with the appropriate link security.

The initial FlightLinux software load is approximately 400,000 bytes in size. The nature of the UoSat-12 memory architecture at boot time limits the load size to less than 512,000 bytes. After the loader, which is read-only memory (ROM)-based, completes, 4 megabytes of memory is available. The software load includes: 1) a routine to setup the environment and 2) a routine to decompress and start the Linux kernel. The kernel is the central portion of the operating system, a monolithic code entry. It controls process management, input/output, the file system, and other features. It provides an executive environment to the application programs, independent of the hardware.

Extensive customization of the SETUP routine, written in assembly language, was required. This routine in its original form relies on BIOS (Basic Input/Output System) calls to discover and configure hardware. In the UoSat-12 configuration, there is no BIOS function, so these sections were replaced with the appropriate code. Sections of SSTL code were added to configure the unique hardware of the UoSat-12 computer. The SETUP routine then configures the processor for entry to Protected Mode and invokes the decompression routine for the kernel. The SETUP routine is approximately 750 bytes in length and represents the custom portion of the code for the UoSat-12 software port. This product is usually referred to as the Board Support Package (BSP). The remaining code is COTS Linux software. This process is the same for any FlightLinux port.

We modified the standard Linux SETUP routine, written in assembly language, to be table-driven. This had the added advantage of addressing the export restriction issues. The breadboard architecture includes an asynchronous serial port for debugging. We used this extensively for debugging the SETUP module. On the spacecraft, the asynchronous port exists, but it is not connected to any additional hardware.

FlightLinux was implemented in an incremental manner. The initial software build does a "Hello, World" aliveness indication via the asynchronous port and allows login. The synchronous serial drivers must be integrated to allow communication in the flight configuration. The bulk memory device driver, which uses the 32-megabyte modules of extended memory as a file system, can be added next. The breadboard has a single 32-megabyte module, and there are four modules in the flight configuration. The CAN bus drivers and the network interface can be added later. Additional modules could be uplinked later on an incremental basis.

We did not complete the testing of the FlightLinux software on the UoSat-12 breadboard within the allocated resources. It is difficult to reasonably estimate the effort required to complete the testing.

Steps to FlightLinux Definition

We defined the steps to a space-flight demonstration of the Linux operating system. Regardless of the implementation architecture, certain pivotal issues must be defined. This was done in a series of reports. These reference reports were archived together in one place along with ongoing research related to the topics. The key issues include: the architecture of the target systems, the nature of application software, the architecture of an onboard LAN, and the requirements for support, the architecture of the onboard storage system, the requirements for support, and the nature and design of the software development testbed.

The Target Architecture Technical Report examines the current, near-term, and projected computer architectures that are and will be used on board spacecraft. The resulting list allows examination of the feasibility and availability of Linux. The choice of the actual architecture for implementation was determined more by opportunity of a flight than by choice of the easiest or most optimum architecture.

The *POSIX Report* examines and documents the POSIX-compliant aspects of Linux and other Flight Operating systems as well as the POSIX-compliant nature of legacy flight application software. This is an ongoing effort by GSFC Code 582, the Flight Software Branch.

The *Onboard LAN Architecture Report* discusses: 1) the physical level interfaces on existing and emerging missions and 2) the device drivers required to support IP over these interfaces. Ongoing work in this area is being done by the Consultative Committee for Space Data Systems (CCSDS) committee and the OMNI Project. The choice of a demonstration flight will define which interfaces will need to be implemented first. In addition, those interfaces with COTS drivers, and those for which device drivers need to be defined were delineated.

The Bulk Memory Device Driver Report will define the approach to be taken to implement the Linux file system in the bulk memory ("tape recorder") of the spacecraft onboard computer. It will define which elements are COTS and which need to be developed.

These reports are living documents and were updated to document new developments. The reports are intended to be stand-alone, but reference the other reports as required. A major purpose of the reports is to collect in one area the COTS aspects of the specific aspect of the FlightLinux implementation so that attention could be focused on the remaining "missing pieces." A summary of each report is presented below.

Target Architectures

Various microprocessor architectures have been and are being adapted from commercial products for space flight use. For all of the primary architectural candidates we identified, Linux is available in COTS form. The primary hardware for flight computers in the near term appear to be derived from the Motorola PowerPC family (RHPPC, RAD6000, RAD750), the SPARC family (EH32), the MIPS family (Mongoose, RH32), the Intel architecture (space flight versions of 80386, 80486, Pentium, Pentium-II, Pentium-III), and the Intel Advanced RISC machine (ARM) architecture. Versions of FlightLinux for the PowerPC and MIPS family are important goals. Because almost all of the effort in developing onboard computers for spacecraft seems to involve adapting existing commercial designs, the logical next step is to adapt COTS software, such as the Linux operating system.

Given the candidate processors identified in missions under development and planned in the short term, we then examined the feasibility of Linux ports for these architectures. In every case, a Linux port was not only feasible, but is probably available COTS. Each needs to be customized to run on the specific hardware architecture configuration of the target board.

Existing space processors in recent or planned use include the RAD6000, the RH32, and the MIPS-derived Mongoose-V. Generally, Linux requires a Memory Management Unit (MMU) for page-level protection, as well as dynamic memory allocation. However, ports of Linux (uCLinux) exist for the Motorola ColdFire processor series and similar architectures, all without memory management. The Mongoose architecture does not include memory management hardware. A Mongoose port of Linux is feasible, and this has been examined in conjunction with GSFC, Code 582, Flight Software Branch. The future usage plans of these hardware architectures determine the direction of our efforts on the FlightLinux software ports.

Emerging space processors include Honeywell's Radiation-hard PowerPC (RHPPC), the Lockheed's RAD750, European Space Agency's (ESA's) ERC32, and the Sandia Laboratories' radiation-hard Pentium. All are viable targets for FlightLinux. The RHPPC and the RAD750 are variations of the Motorola PowerPC architecture. GSFC Code 586 already has Linux running on the PowerPC architecture, in a laboratory environment. The Intel (Pentium) version of Linux is the most common, and can be found in the Code 586 lab as well. European Space Agency's (ESA) ERC32 is a variation on the SPARC architecture, and Linux is available for the Sun Sparc architecture. The term COTS in this context should be taken to mean that a commercial version for that processor architecture is available. A specific port for the flight computer embedded board involves coding specific device drivers, reconfiguration, and recompilation of the kernel. Linux is a 32-bit operating system, appropriate for matching the emerging 32-bit class of flight computers.

POSIX

POSIX is an IEEE standard for a Portable Operating System based Unix. The use of a POSIX-compliant operating system and applications has many benefits for flight software. Among these benefits are: 1) software library reuse between missions and 2) software commonality between ground and flight platforms. For compliant code, the function calls, arguments, and resultant functionality are the same from one operating system to another. Source code does not have to be rewritten to port to another environment. Linux variants are mostly, but not completely, POSIX-compliant. The POSIX standards are now maintained by an arm of the Institute of Electronic and Electrical Engineers (IEEE) called the Portable Applications Standards Committee (PASC) with the associated web site http://www.pasc.org/.

Running a POSIX Test Suite, available from the National Institutes of Standards and Technology (NIST) certifies POSIX compliance. At the moment, we have no plans for POSIX compliance testing of various variations of Linux.

The advantages of Linux are numerous, but the requirements for spacecraft flight software are unique and non-forgiving. Traditional spacecraft onboard software has evolved from being monolithic (without a separable operating system), to using a custom operation system developed from scratch, to using a commercial embedded operating system such as VRTX or VxWorks. None of these approaches have proved ideal. In many cases, the problems involved in the spacecraft environment require access to the source code to debug. This becomes an issue with commercial vendors. Cost is also an issue. When source code is needed for a proprietary operating system, if the manufacturer chooses to release it at all, it is under a very restrictive non-disclosure agreement, and at additional cost. The Linux source is freely available to the team at the beginning of the effort.

As a variation of Linux, and thus Unix, FlightLinux is Open Source, meaning the source code is readily available and free. FlightLinux currently addresses soft real-time requirements and is being extended to address hard real-time requirements for applications such as attitude control. There is a world-wide experience base in writing Linux code that is available to tap.

The use of the FlightLinux operating system simplifies several previously difficult areas in spacecraft onboard software. For example, the FlightLinux system imposes a file system on onboard data storage resources. In the best case, Earth-based support personnel and experimenters may network-mount onboard storage resources to their local file systems. The FlightLinux system both provides a path to migrate applications onboard and enforces a commonality between ground-based and space-based resources.

We have pursued the IEEE POSIX compliance issues of standard embedded Linux, in parallel with an effort in GSFC Code 582, which has collected a library of POSIX-compliant flight applications software. FlightLinux also enables the implementation of the Java Virtual Machine (JVM), allowing for the up-link of Java aplets to the spacecraft.

Linux is not by nature or design a real-time operating system. Spacecraft embedded flight software needs a real-time environment in most cases. However, there are shades of real time, specified by upper limits on interrupt response time and interrupt latency. We can generally collect these into hard real-time and soft real-time categories. Examples of hard real-time requirements are those of attitude control, spacecraft clock maintenance, and telemetry formatting. Examples of soft real-time requirements include thermal control, data logging, and bulk memory scrubbing.

Unix, and Linux, were not designed as real-time operating systems, but do support multi-tasking. Modifications or extensions to support and enforce process prioritization are necessary to apply Linux to the embedded real-time control world.

In one model, a process may yield the central processing unit (CPU) to another pending task. In a preemption scheme, a running process is stopped, and a pending process is started. In another scheme, time slicing, a "round-robin" priority scheme allows equal access to all tasks, or a variation, with a high-priority and a low- priority queue. It is generally agreed that a preemptive scheduling scheme allows for greater concurrency in a real-time system. Beyond the process-switching scheme is the interrupt prioritization. Here, we mean asynchronous interrupts from external sources. Interrupt prioritization is determined and enforced by the hardware configuration. Also, interrupt servicing supersedes software process execution in general.

Problems originate from the fact that the traditional Unix or Linux kernel is a monolithic entity that governs process prioritization. Interrupt drivers and the kernel itself do not participate in the prioritization scheme. The kernel typically has large stretches of non-pre-emptible code. This is necessarily in the design so that data structures can be modified in an atomic fashion. In a Linux kernel, all interrupt handlers run at a higher priority than the highest-priority task. In the Unix view, the kernel is the top level and most important task. In the real-time control world, this is not necessarily true.

One approach to correcting this is to implement a threaded execution approach for the kernel and the interrupt handlers. The question arises as to how much the Linux kernel can be modified and still be referred to as a Linux kernel. Another approach is to treat the kernel itself as a scheduled task, under a Real Time Task Manager that manages process prioritization and takes over control of interrupts. This has been referred to as kernel cohabitation.

Many real-time schedulers for Linux are available for download. These are a Rate Monotonic Scheduler (RMS), which treats tasks with a shorter period as tasks with a higher priority, and an Earliest Deadline First (EDF) scheduler. Other approaches are also possible. It is not clear which approach provides the best approach in the spacecraft-operating environment. This should be investigated and tested. The results of our POSIX investigations of the Linux system and legacy flight application code are documented in Ref. [2].

The Bulk Memory Device Driver

Spacecraft onboard computers do not usually employ rotating magnetic memory for secondary storage. Initially, magnetic tape was used, but now the state of the art is to use large arrays of bulk Dynamic Random Access Memory (DRAM), with various error detection and correction hardware and/or software applied.

A device driver is the low-level software routine that interfaces hardware to the operating system. It abstracts the details of the hardware, in such a way that the operating system can deal with a standardized interface for all devices. In Unix-type operating systems such as Linux, the file system and the I/O devices are treated similarly.

Device drivers are prime candidates for implementation in assembly language, because of the need for bit manipulation and speed. They can also be implemented in higher-order languages such as "c" however. Typical device drivers include those for serial ports, for parallel ports, for the mass storage interface (Small computer system interface (SCSI), for example), for the LAN interface, etc. Device drivers are both operating system-specific, and specific to the device being interfaced. They are custom code, created to adapt and mediate environments.

The current state of the art of spacecraft secondary storage is bulk memory, essentially large blocks of DRAM. This memory, usually still treated as a sequential access device, is mostly used to hold telemetry during periods when ground contact is precluded. Bulk memory is susceptible to errors on read and write, especially in the space environment, and needs multi-layer protection such as triple-modular redundancy (TMR), horizontal and vertical Cyclic Redundancy Codes (CRC), Error Correcting Codes (ECC), and scrubbing. Scrubbing can be done by hardware or software in the background. The other techniques are usually implemented in hardware. With a Memory Management Unit (MMU), even using a one to one mapping of virtual to physical addresses, the MMU can be used to re-map around failed sections of memory.

Although we usually think of bulk memory as a secondary storage device with sequential access, it may be implemented as random access memory within the computer's address space. This is the case with the OBC of UoSat-12.

The Flash File System (FFS) has been developed for Linux to treat collections of flash memory as a disk drive, with an imposed file system. Although we are dealing with DRAM and not flash, we can still gain valuable insight from the FFS implementation. In addition, the implementation of Linux support for the personal computer memory card international association (pcmcia) devices provides a useful model.

The onboard computer on the UoSat-12 spacecraft has 128 megabytes of DRAM bulk memory. It is divided into four banks of 32 megabytes each, mapped through a window at the upper end of the processor's address space. This is the specific device driver that the QSS team develops and uses as a model for future development of similar modules. The current software of the UoSat-12 onboard computer treats this bulk memory as paged random access memory and applied a scrubbing algorithm to counter environmentally induced errors.

The ram disk is a disk-like block device implemented in RAM. This is the correct model for using the bulk memory of the onboard computer as a file system. Multiple RAM disks may be allocated in Linux. The standard Linux utility "mke2fs," which creates a Linux second extended file system, works with RAM disk, and supports redundant arrays of inexpensive disks (RAID) level 0.

The RAID model was developed to use large numbers of commodity disk drives combined into one large, fault-tolerant, storage unit. The approach can be applied to bulk memory as well. RAID can be implemented in software or hardware. For the purposes of this document, we consider RAID software implementations. Software RAID is a standard Linux feature, available as a patch to the 2.12 Kernels and slated to become an included feature in Kernel 2.4.

This initial version of the driver uses memory mirroring, with memory scrubbing techniques applied. In the simplest case, we treat three of the four available 32-megabyte memory pages as a mirrored system. The memory scrubbing technique is derived from the current scheme used by SSTL, as is the paging scheme. The next version of the driver uses all four of the available 32-megabyte memory pages with distributed parity. The performance with respect to write speed is expected to be less than with the Level 0, but the memory resilience with respect to errors is expected to be much better.

It is unclear without further testing whether the RAID technique will be sufficient to counter the environmentally-induced errors expected in the bulk memory on-orbit. It is generally accepted that RAID is not intended to counter data corruption on the media, but rather to allow data recovery in case of media failure. A defined testing approach will be used with the bulk memory device driver on the breadboard facility. More extensive testing on-orbit with the UoSat-12 spacecraft is required to validate the approach. The Bulk Memory Device driver approach that we defined is documented in Ref. [4].

Onboard LAN

Given that the Linux operating system is onboard the spacecraft, support for a spacecraft LAN becomes relatively easy. Extending the onboard LAN to other spacecraft units in a constellation also becomes feasible, as does having the spacecraft operate as an Internet node. For space to ground communication, FlightLinux can utilize the IP-in-space work validated by the OMNI Project.

Interface between spacecraft components is usually provided by point-to-point connections, or a master/slave bus architecture. The use of a LAN onboard is not yet common. This is partially due to the lack of space-qualified components.

The avionics bus MIL-STD-1553 and its optical derivative, 1773, are commonly used between spacecraft components. This bus, used in thousands of military and commercial aircraft has a legacy of applications behind it. Also, 1553 is transformer-coupled and dual-redundant, providing a level of failure protection. The raw data rate is 1 megabit-per-second. It is a master/slave architecture.

For point-to-point connections that do not require the complexity of a 1553/1773 connection, a synchronous serial connection such as RS-422/23 with a bit rate of approximately 1 megabit-per-second is typically used.

A LAN-type architecture is typically used in office and enterprise environments (and spacecraft control centers). It provides a connection between peer units, or clients and servers. The typical LAN uses a coax or twisted pair connection at a transmission rate of 10 megabits per second, a twisted pair connection at 100 megabits per second, or optical at 155 megabits per second, with higher speeds possible.

Usually, a LAN is configured with a repeating hub or a central switch between units. The standard protocol imposed on the physical interface is Transmission Control Protocol /Internet Protocol (TCP/IP), although others are possible (even simultaneously). The TCP/IP protocol has become a favored approach to linking computers around the world. Linux and most other operating environments support the protocol.

The UoSat-12 configuration allows us to exercise the TCP/IP and CAN bus components of an onboard LAN. Evolving physical layer interfaces for use onboard the spacecraft include 100 megabit Ethernet and Firewire (IEEE-1394). Although 1553 device drivers exist for Linux, they only allow the use of the legacy bus in the classical master-slave architecture. Ongoing work by the Spacecraft Onboard Interfaces (SOIF) group within the CCSDS is defining the application of IP-over-1553, and, more generally, IP over a master-slave architecture.

Code Integration and Testing

An initial build of the FlightLinux software has been running since March 2001. The UoSat-12 breadboard has been available at the OMNI Lab since April 2001. The breadboard has an associated Windows-NT machine to load software via the CAN bus or the asynchronous port and to provide debugging visibility. From our facility at QSS, we can access the breadboard facility via the "PC-Anywhere" software with the appropriate link security. The following sections give background information on some of the system components.

80386EX Processors

The 80386EX model includes the memory management features of the baseline 80386, along with an interrupt controller, a watchdog timer, synchronous/asynchronous serial I/O, DMA control, parallel I/O and dynamic memory refresh control. These devices are DOS-compatible in the sense that their I/O addresses, direct memory access (dma) and interrupt assignments correspond with an IBM PC board-level architecture. The DMA controller is, however, an enhanced superset of the Intel 8237A DMA controller. The 80386EX processor core is static meaning that the clock can be slowed or stopped without loss of state.

The 80386EX includes two DMA channels, three channels of Intel 8254 compatible timer/counter, dual Intel 8259A interrupt controller functionality, a full-duplex synchronous serial I/O channel, two channels of Intel 8250A asynchronous serial I/O, a watchdog timer, 24 lines of parallel I/O, and support for dram refresh. The 80386EX can interface with the 80387 math co-processor, and the SSTL breadboard is equipped with an 80387SL.

Specific Device Drivers

A large number of device drives for custom I/O interfaces are available in open-source form for Linux. The physical-level I/O interfaces most likely to be required include asynchronous serial, synchronous serial, 1553/1773, CAN (ISO 11898), and Ethernet (IEEE 802.3 10-base-T). The device driver needs to be customized for the specific physical layer hardware used to implement the interface. We have identified existing software drivers for all of these. The UoSat-12 OBC uses asynchronous serial (debug only) synchronous serial, 10Base-T, and CAN. CAN and 10Base-T drivers are COTS, and the OMNI group has developed a synchronous serial HDLC device driver that we can use.

FlightLinux-specific Kernel Enhancements or Application Code

Certain extensions to a standard Linux kernel, beyond an embedded version, will have to be made for the unique space flight environment. These enhancements may take the form of application code, running under the kernel; this issue is currently "to be decided." Such enhancements will include a bulk memory device driver/file system (using bulk memory as a file system - to be discussed in a subsequent report), a memory scrub routine to periodically check and correct memory for radiation-induced errors, and a watchdog timer reset routine, to detect radiation-induced latchup of the processor.

Non-BIOS Systems

The UoSat-12 80386EX embedded board does not contain a BIOS, the firmware-based basic I/O system that is common in desktop computers. The functions that a BIOS provides include power-on self test, hardware configuration establishment, and software environment establishment. The UoSat-12 board does have a 32-kilobyte ROM that contains the Loader code. Two versions are available: one for loading from the asynchronous serial maintenance port and another for loading via the CAN bus. The loader code includes an Initialization routine, which provides some of the functionality of the BIOS in the sense of configuring the hardware and software environment. Run time support to programs is not provided. After the code is loaded, the ROM is mapped out of the memory space.

The Linux system in general, at least the 80x86-based implementations, rely on a BIOS for the initial system load. When a BIOS is not present, as in an embedded system, the functionality must be provided by other means. Once the Linux kernel is up and running, Linux does not rely on the BIOS. Architectures other than the 80x86 contain firmware that may be broadly classified as a "BIOS."

Embedded Debugging Tools

Because the embedded system does not have the usual human interfaces such as keyboard and screen, alternative approaches must be found for debugging. One of these is the gnu debugger (gdb). When the target system (in this case, the SSTL 80386EX board) does not have a console, the gdb will run remotely on the host, with several minimal modules in the target and a link via the serial port. We implemented the gdb on the UoSat-12 breadboard. UoSat-12 Breadboard debugging options we explored include:

1. BlueCat Vendor Support - We started with the idea of using the BlueCat release of Linux from LynuxWorks, and we have copies of Version 2 and 3. We are currently using the ELKS distribution of Linux, because it is very simplified, and can be brought up, for example in real mode on the 80386; it does not require protected mode, with its associated complexities. The idea was to get something working with ELKS, then switch to BlueCat. One approach is to upgrade to the latest (version 4) release of BlueCat Linux, and get the associated priority support option. This would make available to us a BlueCat application engineer. The cost of this option was not within the budget. The Omni lab also currently uses BlueCat, but does not have a copy of Version 4. BlueCatRT - the real time extension, was announced on June 10, 2002.

2. ICE - an in-circuit emulator, uses a pod to replace the central processing unit (cpu), that cables to a box. A 80386EX ICE unit was not available at GSFC for loan. They are very expensive pieces of equipment. This turns out not to be an option with the UoSat-12 breadboard, as the cpu chip is soldered in. The UoSat unit is more of a proto-flight unit than a breadboard. It is not conformally coated.

3. Logic Analyzer - There is a logic analyzer available in the Omni Lab. This unit can be considered a generic ICE - it can capture and display logic signals, can be triggered by predefined events, and can buffer a series of states on logic line. It would normally be used on the cpu's address and data bus, and some selected control signals. It is not processor-specific. There is a learning curve in its use, and the operator must understand software-hardware interactions.

4. Rom-based Monitor. On the UoSat-12 board, the ROM based monitor contains the load and dump code, and rudimentary hardware set-up routines. It is NOT a BIOS. We have the ability to plug a ROM emulator into the ROM socket, or to program custom proms. This would require an external prom programmer unit. A much better solution is to have the firmware in a flash-rom, which can then be modified in-circuit. But, this option has to be included in the board design, and is not a feature of the UoSat-12 OBC. With a large enough flash rom, significant portions of the Linux kernel could be included as well. This is the approach to take if the hardware can be specified.

5. A rom emulator plugs into the rom's socket, replacing the rom. This is also sometimes referred to as a Prom-ICE. Since the prom is replaced by ram on the debugging system, many options are possible. We did not located such a unit for the UoSat.

The Embedded Testbed Report (ref. 5) gives additional details on these facilities.

TRL Assessment

This Project started with an estimated TRL level of 3. We progressed to a TRL level estimated at greater than 5. Our goal was TRL level 7. The steps required to progress from the current state to TRL Level 7 are as follows:

1. Debug the FlightLinux load on the UoSat breadboard, level of effort tbd

2. Validate the approach to software export. Level of effort, tbd.

3. Have SSTL download the software and test it on their breadboard facility. Jointly develop an on-orbit test plan and associated procedures. This should take several weeks.

4. Uplink the software to the UoSat-12 Spacecraft and verify the load. This should take about 1 day.

5. Execute the test plan. This should take about 2 weeks, although the testing period may be open ended, if the software is left in place.

Milestones

At the inception of this project, we defined 15 milestones. Here is a discussion of the status of those milestones at the end of the project. We did not define a milestone for code implementation. Where we found that the effort was being pursued by another group, we did not duplicate that effort, but chose to cooperate and collaborate with those efforts. This allowed us to apply more resources to code testing.

1. "The first thing we will do is set up a web site to disseminate timely information about the FlightLinux project, and to serve as a central information repository. This website will serve as the key point of contact for progress milestones and product features."

The website has been up since the project was initiated, and has proven an excellent outreach tool. We upgraded the site per current NASA/GSFC security and usability guidelines. The website is discussed further in the Outreach section.

2. "Investigate POSIX real-time operating systems (VxWorks, RT-Linux, LynxOS, etc) for spacecraft onboard use. This effort will be coordinated with a similar ongoing effort in Code 582. This work will result in a detailed report on this topic."

We worked closely with Code 582 on this. They are building a library of legacy flight application code that is Posix compliant, and guidelines for development of Posix-compliant application code for flight application. We participated in a study with them on the level of Posix compliance of various flight operating systems.

3. "Investigate Selected Target Architecture(s) for FlightLinux. This tradeoff study will investigate the trends in the emerging hardware platforms for spacecraft computer use, and decide where the center of mass resides. This work will result in a detailed report on this topic, which will serve as guidance for the target architecture for the software development work."

The selected target architecture is the 80386EX, due to the availability of the UoSat-12 spacecraft in the proper time frame. The Target Architecture Report discusses the various options.

4. "Configure POSIX development workstation for flight code. This effort will integrate a desktop workstation to development and manage code for the selected target architecture. This will include setting up an FTP archive of software. The workstation will be selected from those already available in Science Data Systems laboratory."

Workstations for code development have been up and working since project inception. We have accumulated a large library of software.

5. "Configure embedded testbed for flight code testing. After the target architecture is selected, a testbed environment for that architecture will be configured for integration and test. This will make use of existing resources in the Flight Software Systems Lab."

The testbed being used is the UoSat-12 breadboard, under the auspices of the OMNI project. We also have embedded testbeds at QSS. The differences between the testbed configurations are discussed in an appendix.

6. "Design, develop and implement a bulk memory device driver under Linux. This effort will develop a Linux device driver for "typical" bulk memory devices typically used as tape recorder replacements. This software will abstract the bulk memory into a block oriented file system. This work will result in a detailed report on this topic"

The defined bulk memory device driver is defined. It uses the existing Linux M-Disk and RAID code to implement a file system in the bulk memory. It has been tested on the breadboard at QSS.

7. "Investigate architectures for onboard LANs to interconnect instruments and the C&DH. This work will result in a detailed report on this topic, which defines the concept of the onboard LAN for science data."

We are focused on the LAN interfaces of the UoSat-12, which include synchronous-serial (point-point for T&C), 10-Base-T, and CAN bus. We chose to defer consideration of IP-over-1553, which is being defined by the SOIF group. We track their efforts. A 1553 device driver for Linux is available, which allows use of the bus in a classical master-slave mode.

8. "Investigate security issues in the use of FlightLinux, including privacy and mission success criteria. This will correspond with NASA Standards for data processing systems security, and will ensure that the flight data system does not compromise the mission, or allow unauthorized access to data or commanding. This work will result in a detailed report on this topic"

A detailed report was recently release by NASA code 297 and 580. It is the IP-in-Space Security Handbook, dated 9/2001. We coordinated with the authors of this report, and the OMNI project.

9. "Investigate the feasibility of a 1553 bus device driver under Linux. The 1553 bus is a common physical interconnect mechanism used onboard spacecraft (and avionics systems) to interconnect devices. This effort will attempt to develop an IP over 1553 software module. The advantage of this is that it enables the use on an onboard LAN to interconnect the C&DH and instruments. The difficulty will be that the 1553 bus, with bus master and remote terminal concept, does not lend itself easily to the IP concept. However, the Token Ring and PPP concept may apply. Given the IP-over-1553 software module, the existing Essential Services Node could then be used a network node. If feasible within the existing resources, develop, integrate, and test the device driver."

We have been ignoring 1553, because UoSat-12 does not use it. However, we have been closely tracking the work of the SOIF group, who are defining an IP-over-

1553 standard. A 1553 device driver, which allows Linux to use the 1553 bus in a master-slave mode is COTS.

10. "Investigate IP over 1394 (FireWire) and other carrier mechanisms. IP over 1394 bus has been the topic of an Internet Engineering Task Force study since 1997. This study will take a look at emerging transport technologies that could replace 1553 in radiation hardened, low-power versions for spaceflight use. This work will result in a detailed report on this topic."

This material is included in the Onboard LAN architecture report. IP over 1394 is being implemented at GSFC by the NPP Project. Linux device driver support for 1394 is cots.

11. "Develop a cost/benefit analysis of producing a flight load for WIRE on-orbit Testbed…"

The cost/benefit analysis did not favor the use of WIRE, and we abandoned that track when the UoSat-12 became available. We recently revisited the decision, and came to the same conclusion.

12. "Evaluate/assess legacy (MAP, WIRE, SMEX-lite, etc) Flight Software for rehosting under FlightLinux. This effort will be conducted in conjunction with the Flight Software group (code 582), and will examine and collect libraries of application Flight code to recompile under POSIX standards, and for the selected target architecture."

This work was done by GSFC Code 582, and showed that POSIX-compatible versions of legacy flight code was feasible and did not involve extra effort.

13. "Produce *Lessons Learned* Report(s) on converting existing flight software suites to Linux. This report, also to be hosted on the website, will document the process of developing flight code for the FlightLinux environment. It will cover both the adaptation of existing legacy code, and the guidelines for new code. It will identify any technology dependencies of the FlightLinux, to transition to TRL 7."

The Lessons Learned report was issued on 6/14/2002, and the material is included in the Lessons Learned section of this report.

14. "Determine draft Coding Standards and API's for POSIX-compliant Flight Software. This effort, also closely coordinated with the Flight Software group, will collect a set of coding standards for developing applications and application program interfaces for FlightLinux."

This effort was done by GSFC Code 582.

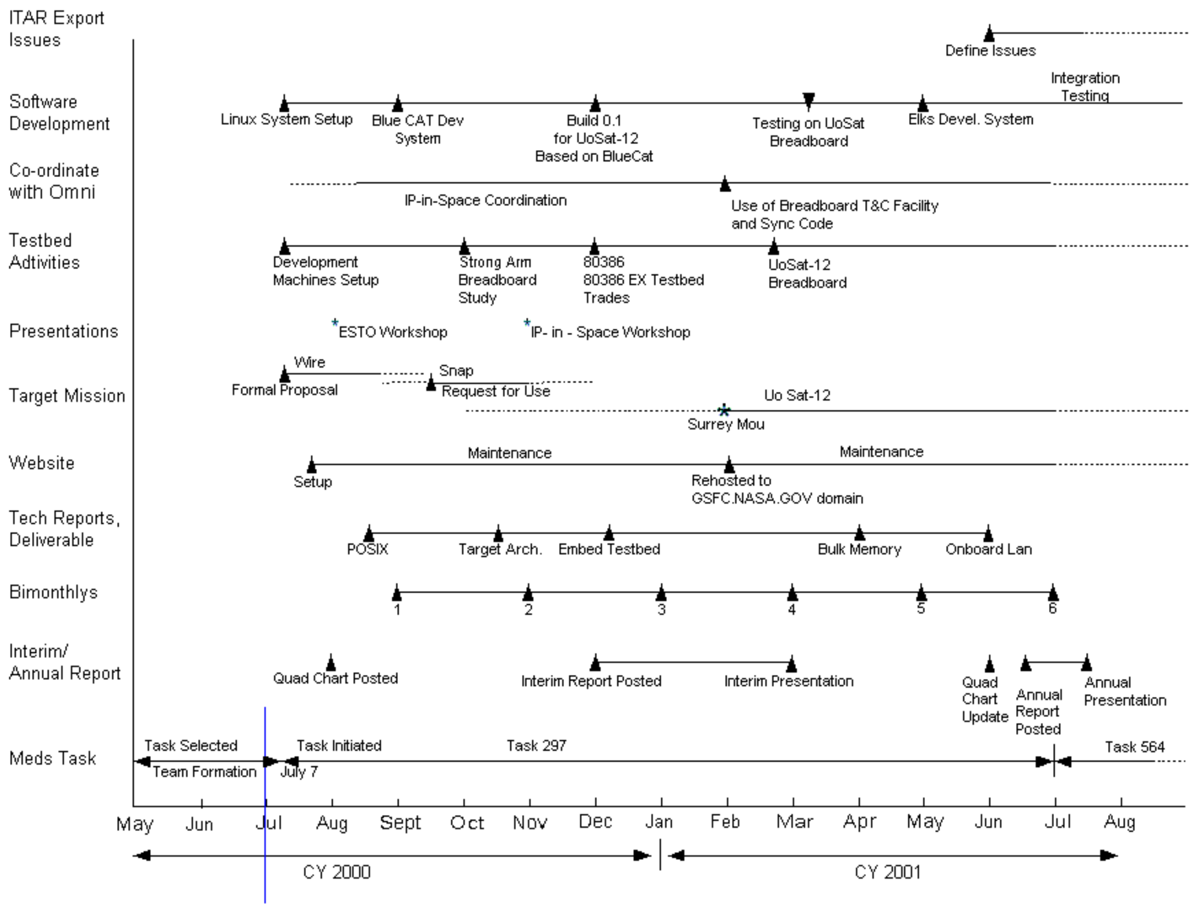
15. "Report on trends in Flight Hardware, Flight Operating Systems, and Flight Software. This task represents a continuing effort to remain cognizant on the state of the art in the

hardware and software environments for onboard use. This information will be kept on the website, and presented formally at least yearly. This will cover GSFC missions, other NASA missions, ESA and NASDA missions, commercial practice, and whatever other information we can collect."

We used the website, which was kept continually up-to-date, to hold and point to information on Flight Hardware, Flight Software, trends in technology, and other flight project status.

Schedule

The proposed schedule for the first task year of the FlightLinux Project is presented on the next page. Most of the second task year was dedicated to code integration and testing, and keeping the various reports current. The StrongArm processor breadboard study was conducted, based on the possibility of using the SNAP spacecraft. The results were incorporated in the Embedded Testbed Report.


ITAR Export Issues
Define Issues
Software Development
Linux System Setup
Blue CAT Dev System
Build 0.1 for UoSat-12 Based on BlueCat
Testing on UoSat Breadboard
Elks Devel. System
Integration Testing
Co-ordinate with Omni
IP-in-Space Coordination
Use of Breadboard T&C Facility and Sync Code
Testbed Adtivities
Development Machines Setup
Strong Arm Breadboard Study
80386 80386 EX Testbed Trades
UoSat-12 Breadboard
Presentations
ESTO Workshop
IP- in - Space Workshop
Target Mission
Wire
Formal Proposal
Snap
Request for Use
Uo Sat-12
Surrey Mou
Website
Setup
Maintenance
Rehosted to GSFC.NASA.GOV domain
Maintenance
Tech Reports, Deliverable
POSIX
Target Arch.
Embed Testbed
Bulk Memory
Onboard Lan
Bimonthlys
1
2
3
4
5
6
Interim/ Annual Report
Quad Chart Posted
Interim Report Posted
Interim Presentation
Quad Chart Update
Annual Report Posted
Annual Presentation
Meds Task
Task Selected
Team Formation
Task Initiated
July 7
Task 297
Task 564
May
Jun
Jul
Aug
Sept
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
CY 2000
CY 2001

Lessons Learned

1. **Developing software for an embedded system is always harder than you imagine, even when you allow for past lessons learned**. The UoSat-12 computer was a custom design by SSTL. It is based on the now-unavailable Intel 80386EX processor, and is built to be very close to a pc architecture. However, it does not have a BIOS (a firmware-based part of the operating system). We were unable to located any debugging hardware or software at GSFC after extensive search. We were also unable to locate any specific COTS hardware or software for debugging still available.

The normal boot process for the Linux system involves the BIOS invocation of the SETUP routine, which is coded in assembly language, and prepares data tables for the kernel. After SETUP finishes its job, control is transferred to the kernel code. In the UoSat-12 case, because there was no BIOS code per se, we used the SSTL loader to load the SETUP and kernel code, and transfer control to SETUP. The loader code, contained in a PROM on the UoSat-12 breadboard, did some hardware configuration, but did not include the functionality of a BIOS.

After we debugged the SETUP routine, control was transferred into the Linux Kernel. At this point we were in the netherworld with no visibility of processes, before the kernel's serial console had not yet been configured. We could not make use of the gdb tool (gnu debugger), because it relies on having the kernel serial port available. We did not have a hardware CPU probe available, although the OMNI Project offered us the use of a logic analyzer. At the end of the available funds, we did not have the path through the kernel code working or debugged. This may have been due to a configuration parameter we missed in SETUP, or a memory configuration issue (an artifact of the loader). We employed extensive code reviews and traces, but were not able to isolate the problem within the resources available.

On the positive side, using the breadboard remotely at GSFC from our facility about a mile away worked very well. In worst case, we could drive over to the breadboard in a matter of minutes, but this was rarely necessary. It was much easier to use than if the facility had been located in the UK.

If we had to start this project over with our current knowledge, we would more carefully assess the support systems available for the target architecture. We chose the UoSat-12 system more due to availability than a rigorous examination of suitability. To have a chance of achieving a Technology Readiness Level (TRL) of 7, on-orbit testing, we had to find a project that was already on-orbit, or close to launch. This severely limited our decision space. In retrospect, two years was probably too short for this project. Alternately, we might have chosen to stop at TRL level 5, and find a Flight project to collaborate with for further efforts.

2. **Exporting Satellite Control Software is a big deal.** As we abruptly learned about 1 year into the project, the International Trafficking in Arms (ITAR) regulations apply to any satellite control software, of which FlightLinux, as the onboard operating system,

would be a part. ITAR regulations would require a review and certification by GSFC, NASA, Department of Defense (DoD), Department of State (DOS), and other agencies, a very time-consuming process.

Initially, we would have been faced with the review of perhaps a half-million lines of code for the Linux system. However, we realized that all of that was already in the public domain and available for download internationally. The only unique part would be the SETUP routine. (The configuration of the kernel, a key part of the FlightLinux load, involves the choice of standard modules and device drivers to include and exclude form the software build.). Given this, we reduced the unique problem to several hundred lines of code.

We modified the standard Linux SETUP routine, written in assembly language, to be table-driven. This had the advantage of addressing the export restriction issues, while making the SETUP routine more generic. Our proposed approach, which has not been validated by legal or export-control authorities, is that the generic version of the SETUP code, which we released back into the public domain under the terms of the gnu license, is not an issue. The content of the table, which is data, not code, is the critical aspect.

We filed a form 1679, a NASA "Disclosure of Invention and New Technology (including Software)" with the GSFC Patent Counsel. On July 20, 2001, we answered the "GSFC Global Concerns Statement." We also prepared the "GSFC Software Public Disclosure Export Control Checklist." It must be emphasized that our proposed approach to the export control issue has not been validated, and there may be further stumbling blocks to be encountered. However, other Linux based software, such as Beowulf, has been through the process successfully.

Reference: emails

Subject: Re: policy or position
Date: Thu, 17 May 2001 15:51:40 -0400
From: Lee Holcomb <lholcomb@hq.nasa.gov>
To: Pat Stakem <pstakem@qssmeds.com>

Pat:

Outstanding questions and efforts. I have in fact spoken to Ed Frankel our agency general council on this very topic. It was a result of the effort we had to go thru to get a release authority for the Beowulf software that caused the discussion with Code G. Last year the Presidents IT Advisory Council studied the same subject. I have a keen interest in developing an effective policy that allows the government to both use open source (which includes the GNU requirement of publication) and the requirement for export clearance. Let's discuss this more fully.

Lee
--

Lee Holcomb
NASA Chief Information Officer
(202) 358-1824

Subject: Re: policy or position
Date: Thu, 17 May 2001 17:01:37 -0400
From: Lee Holcomb <lholcomb@hq.nasa.gov>
To: Pat Stakem <pstakem@qssmeds.com>

Your points are mostly on target. The export issue is an issue because you are developing satellite computing capability which is on the State Department Munitions list. I realize that this is in direct conflict with the GNU policy. I believe that the government needs to take into account the free software issues. That said, the best answer is that we are formulating our policy. I believe we can give you a specific determination for your project. I suggest you and I begin a dialog on this issue and I will bring in general council.

Lee
--
Lee Holcomb
NASA Chief Information Officer
(202) 358-1824

Subject: Re: policy or position
Date Fri, 18 May 2001 16:07:51 -0400
From: Lee Holcomb <lholcomb@hq.nasa.gov>
To: Pat Stakem <pstakem@qssmeds.com>

Pat:

I have jumped into this issue with some comments, but not enough time to really work your issues to ground. I would suggest you work internal at GSFC to make sure that my comment on the need for an export lic is required. If you are operating on a government to government MOU, this may serve to allow the export in this unique case. Again, this is an issue for legal council and patent /export people at GSFC to advise you. I would be interested in the outcome.

Lee
--
Lee Holcomb
NASA Chief Information Officer
(202) 358-1824

3. **Outreach works**. Our main approach to public outreach has been the project website, which has been up since the beginning.

Since we posted our goals of keeping the FlightLinux open source, within the meaning of the GNU license, we have had numerous offers of collaboration on the project. These include representatives of worldwide aerospace companies, and individuals. The interest in the FlightLinux Project is growing, due to increasing exposure of the website. We are aware of a dozen flight projects using Linux. Many individuals contacted us for additional information, or to volunteer to work on the project. We declined these offers, as we did not have an approach to include their efforts. However, the level of interest in this project was very high, and world-wide, attracting very high levels of talent to the possibility of contributing to a space-based application. Most notably, Richard Stallman, the originator of the Free Software movement, conducted an extended dialog with us.

A recent Internet search found over 800 references to the FlightLinux Project in 23 countries (Russia, Spain, Mexico, Germany, UK, Italy, France, Greece, Argentina, Netherlands, Czech Republic, Finland, China, Taiwan, Poland, Denmark, Norway, Brazil, Hungary, Canada, Bulgaria, Romania, and Belgium). Following is a list of other Linux in Space Efforts we have corresponded with. In a sense, our evangelical efforts have paid off, and many projects are now considering and evaluating the use of Linux in the onboard environment.

Here is a list of other Linux in Space Efforts we have corresponded with. There is also a list of selected Project correspondence in an Appendix, gleaned from the 800 or so email messages received during the conduct of this effort.

Linux in Space efforts			June 2002	
Group	**Country**	**Mission**	**cpu**	**status**
GSFC	USA	ST-7	PPC-750	paper study
ASRI	Australia	JAESAT	ARM	in development
Hacettepe U.	Turkey	UPESAT	8086	in development
QinetiQ (DERA)	UK	STRV	ERC-32	in work
Montana State U.	USA	sounding rocket	Pentium	flight, 2004
ITT	USA	next gen s/c	PPC-750	Prototype in 2003
JPL	USA	Europa Orbiter	PPC-750	alt. to VxWorks
nsbf	Italy	balloon	80386	in work
Honeywell Space	USA	-?-	PPC-750	-?-
U. Michigan	USA	Mars Rover	-?-	prototype
Navy Postgrad	USA	various	8086	in work
U. NSW	Australia	BlueSat	ARM	In work

OMNI/GSFC	US	CANDOS	80686	STS-107

Notes on the Projects in the table:

1. The in-house ST-7 Study by GSFC chose Linux for the onboard operating system. In their words, "Real-time Linux and VX-Works were both examined as potential candidates for an Operating System (OS) for the ST7 spacecraft. While each OS offers its own unique set of advantages, Real-time Linux was chosen to support the ST7 Payloads for a number of reasons." Contact is Richard Schnurr, GSFC Code 560, 301-286-6069.

2. ASRI, contact is Geoffrey O'Callaghan, gocallag@au1.ibm.com. Student microsatellite project. http://www.asri.org.au/

3. Hacettepe U. contact Sefer Bora, bora@lisesivdin.net. Undergraduate Physics satellite project.

4. QuinetiQ - a spin-off of DERA in the U.K. contact is Malcolm Appleton, MAPPLETON@qinetiq.com. We have a MOU with QuinetiQ to port FlightLinux to their STRV1d engineering model.

5. Montana State U. contact is Charles Kankelborg, kankel@icarus.physics.montana.edu. This is a NASA funded sounding rocket. Their description of the project is " Flight from White Sands Missile Range is tentatively spring, 2004. We are planning an internal PDR for February 2002. We intend to fly a ruggedized Pentium single-board computer for control and data handling. We're expecting roughly half a gigabyte of data from our solar imaging spectrograph during a 5-minute data taking window. Data acquisition is via one or two DMA cards (TBD), with a TBD interface to telemetry. All data will be stored to flash disk, and as much will be downlinked via telemetry as possible. Control functionality is limited to some simple commanding of cameras and shutters via serial lines.

6. ITT - contact is Chris Langford, Chris.Langford@itt.com

7. JPL - Europa Orbiter. Contacts are Kenny Meyer, Kenny.Meyer@jpl.nasa.gov, and Len Day, len.day@jpl.nasa.gov.

8. NSBF - Italian National Research Council, contact is Dr. Enzo Pascale, pascale@iroe.fi.cnr.it.

9. Honeywell Space, contact is Douglas Jerome, jerome@mate2000.az76.honeywell.com.

10. University of Michigan, Mars Rover Project, contact is Marius Aamodt Eriksen, marius@umich.edu.

11. Naval Postgrad School - contact is Jim Horning, jahornin@nps.navy.mil.

12. University of New South Wales, Australia. Contact David Lee, d.lee_bluesat@lycos.com. Undergraduate microsat project, due for launch in 2004.

13. OMNI Project, GSFC Code 588, CANDOS attached shuttle payload on STS-107, launch in July 2002. Point of contact is James Rash. Uses a pc-104 form factor 80686 cpu, with RedHat Linux 6.1, and a custom HDLC device driver, written in-house.

4. **Partnering and collaborating work**.

The work was conducted by a team from QSS Group, Inc., in conjunction with NASA/GSFC Codes 586 (Science Data Systems), 582 (Flight Software), and 588 (Advanced Architectures and Automation). The government-industry team approach worked smoothly, and we cooperating and collaborated with various projects, most notably, the OMNI Project in Code 588. By the nature of the mutual interests, code sharing within the spirit of the gnu public license was possible. Resource sharing also was commonplace. The OMNI Project hosted the UoSat-12 breadboard that we used for development and test. It is possible for a government-industry team to work together successfully towards common goals, with diminished parochialism and territorial behavior. If we started this project today with our current knowledge, we would choose to work with the same partners.

The spirit of the Free Software movement is based on shared development and debugging This approach has been shown to work in many cases (Linux, the OpenOffice Suite, Mozilla Browser, the gnu tools). We were able to find many of the software tools and elements we needed in the Internet community and at GSFC that we then used to build on. Examples of these include the IP-in-Space Security approach, the HDLC driver (Omni Project), device drivers for the CAN and 1553 buses, the RAID and ramdisk device drivers, and the library of Posix-compliant flight application software by Code 582. Most importantly, we were able to discuss issues synergistically with other Linux developers at GSFC and throughout the world.

Recommendations

1. **Investigate and define an approach to allow and manage Open Source software development for Mission Critical NASA Projects**.

The FlightLinux Project explored new issues in the use of "free software" and open-source code, in a mission critical application. Open-source code, as an alternative to proprietary software has advantages and disadvantages. The chief advantage is the availability of the source code, with which a competent programming team can develop and debug applications, even those with tricky timing relationships. The Open-Source code available today for Linux supports international and ad hoc standards. The use of a standards-based architecture has been shown to facilitate functional integration. It is a misconception that "free software" is necessarily available for little or no cost. The "free" part refers to the freedom to modify the source code.

A disadvantage of developing with Open Source may be the perception that freely downloadable source code might not be mature or trustworthy. Countering this argument is the growing experience that the Open-Source offerings are as good as, and sometimes better than the equivalent commercial products. What is needed, however, is a strong configuration control mechanism. For the FlightLinux product, the FlightLinux Team assumed the responsibility of making and maintaining the "official" version.

Many issues on the development and use of Open-Source software on government-funded and mission-critical applications are still to be explored. Debates continue in the media about the relative merits of Open Source versus proprietary software with respect to security issues. This is an area that is both timely and critical, and will need to result in a definition of policy and preferred approach, as well as guidelines and standards for projects.

2. **Investigate FlightBeowulf**

Having a Linux system enables a plethora of software that runs under Linux. One package is the GSFC-developed Beowulf software which implements a parallel processor on a cluster of Linux machines. This may use several machines collocated in one satellite, or multiple satellites in a constellation linking their computational resources via inter-satellite communications. We have experience in hosting science data processing algorithms on standard Beowulf clusters.

An approach to increase computational speed by the use of a multiprocessor system, dubbed a “Flight Beowulf,” would involve the case where an original problem (application) is split into parts (processes) that are acted upon by separate processors simultaneously. Ideally, given ‘n’ processors, the increased computational speed would be almost ‘n’ times that of a single processor. At any given point in the technology maturity curve, we can apply this technique to multiply the onboard processing power available. Of course, we must be observant of the impacts to other mission parameters such as power, weight, heat production, and size.

This approach has been implemented and validated numerous times in a ground-based environment. This lays the cornerstone for space-based cluster computing with off-the-shelf radiation-hardened processors by replacing the commercial processors in the engineering test unit with the equivalent radiation-hard versions. Such a flight computer has the raw processing power capable of managing a complex scenario of sophisticated event detection and data mining algorithms for multiple instruments in a real-time, on-board environment. This concept would be tested fairly easily in a FlatSat-like environment. (FlatSat is the Omni breadboard system, a PC-104-based, onboard-like architecture.)

3. **Maintain the FlightLinux website**

The website should be maintained as a point of presence, a contact point, and to provide continuity with potential follow-on efforts. The website is currently hosted on a Code 588 server. However, no plans or resources are currently in place to include any new or updated information on the site. Security and routine maintenance will be done by Code 588.

4. **Vigorously pursue follow-on efforts**

FlightLinux is a good idea, and has generated a lot of support and interest. We intend to pursue the feasibility of extending this concept commercially in-house, a FlightLinux.com effort. In addition, we are vigorously pursuing flight opportunities. The fact that we were unable to port the software to a particular unique hardware architecture within the timeframe of the project should not be taken as a negative view of the FlightLinux concept.

References

1. HTTP://FlightLinux.gsfc.nasa.gov/docs/Target_Arch_Report.pdf

In the *Target Architecture Technical Report*, we examine the current, near term, and projected computer architectures that are and will be used onboard spacecraft. From this list, we examined the feasibility and availability of Linux. The choice of the actual architecture for implementation was determined more by opportunity of a flight than by choice of the easiest or most optimum architecture.

2. HTTP://FlightLinux.gsfc.nasa.gov/docs/POSIX.pdf

The *POSIX Report* examines and documents the POSIX-compliant aspects of Linux and other Flight Operating systems as well as the POSIX-compliant nature of legacy flight application software. This is an ongoing effort by GSFC Code 582, the Flight Software Branch.

3. HTTP://FlightLinux.gsfc.nasa.gov/docs/onboard_lan.pdf

The *Onboard LAN Architecture Report* discusses the physical-level interfaces on existing and emerging missions as well as the device drivers required to support Internet Protocol (IP) over these interfaces. The CCSDS committee and the OMNI Project (GSFC, Code 588) are doing ongoing work in this area. The choice of a demonstration flight defines which interfaces will need to be implemented first. In addition, those interfaces with COTS drivers and those for which device drivers need to be defined are delineated.

4. HTTP://FlightLinux.gsfc.nasa.gov/docs/ Bulk_Memory_Device_Driver.pdf

The Bulk Memory Device Driver Report defines the approach to be taken to implement the Linux file system in the bulk memory ("tape recorder") of the spacecraft onboard computer. It defines which elements are COTS and which need to be developed.

5. HTTP://FlightLinux.gsfc.nasa.gov/docs//Embedded_testbed.pdf

The Embedded Test Bed Report defines the requirements and architecture for the facility to develop and validate the operating system code for the flight experiment. Guidance has been drawn from similar past facilities.

6. Linux usage on the Space Shuttle:
http://www.faho.rwth-aachen.de/%7Ematthi/linux/LinuxInSpace.html

7. Linux usage on the International Space Station Freedom
http://www.sheflug.co.uk/featuresoft.htm
http://www.linuxjournal.com/article.php?sid=3024

8. http://www.newsforge.com/article.pl?sid=01/03/13/2112221

9. Sandred, Jon, "Managing Open Source Projects, " NY: John Wiley & Sons, Inc. 2001.

10. Raymond, Eric, "The Cathedral and the Bazaar," O'Reilly & Associates, 1999. also, http://tuxedo.org/~esr/writings/cathedral-bazaar/

11. Autonomy Technology and Onboard Processing Study, Space Technology 7 - Autonomy and Autonomy Pilot Project, NASA/GSFC, 8 February 2002.

<u>Appendix - selected email correspondence on FlightLinux</u>

1. Subject: FlightLinux
Date: Mon, 29 Apr 2002 03:35:10 -0700 (PDT)
From: Andrew Cawthorne <a_cawthorne@yahoo.com>
To: pstakem@qssmeds.com

Hi there,

I am a postgraduate student in the UK doing a thesis on concepts for on-board software on satellites. Currently I am just trying to get on overview of different RTOS being used for this purpose in order to do a comparative study. It would be very helpful to me if you could briefly answer some questions about FlightLinux.

What is the current state of FlightLinux? I heard that It was not being used on UosSat-12 after all. Does it have hard real-time support? What languages/compilers are currently available? What processors are supported? Does it have virtual memory? Does it have memory protection? Where do you see it going in the future?

Any help at all would be much appreciated.

Thank you,
Andrew Cawthorne
(Cranfield University, UK)

2. Subject: FlightLinux "contrib"
Date: Fri, 27 Apr 2001 10:46:12 +0100
From: arnaud.lecuyot@astrium-space.com
To: pstakem@qssmeds.com

Dear Sir,

My name is Arnaud LECUYOT and I work for Astrium Ltd (First European Spacecraft manufacturer). I was really interested to find out about the FlightLinux project. I would like to know what is the current status, and also how do you perceive the "Open Source" side of FlightLinux as regards the fact that it is NASA software. In particular, do you plan to accept contributions from other programmers (space related of course), and what is your position wrt. the availability of the software?

Thanks in advance,
Regards,

Arnaud LECUYOT.

****
Arnaud LECUYOT

Spacecraft Systems Engineer

ASTRIUM Ltd
Gunnels Wood Road - STEVENAGE SG1 2AS - UK
Tel :+44 (0)143877-4357 Fax -8913
Mobile:+44 (0)7720840043
arnaud.lecuyot@astrium-space.com
****

3. Subject: logging behavior and variables and other ideas for identifying onboard software problems
Date: Mon, 17 Sep 2001 19:46:51 +0200
From: <Colin-Paul.Gloster@esa.int>
To: Pat Stakem <pstakem@qssmeds.com>

Dear Pat,

Thank you for your emails to my Colin_Paul_Gloster@ACM.org account last week.

My work at ESA is to do with onboard software maintenance. I do not want to come across as trying to get foreign taxpayers to do my work for me, but would you be able to share ideas or experiences on how software may be realised to come up with useful metrics for keeping care of satellites/probes? Such as re-configurability to the extent that telecommands from the ground can select different parameters to monitor instead of just having a handful of vital targets hard-coded? Or other concepts which could be of use AND be practically implementable? Not just strategies to follow right as an anomaly happens, but observations which could be conducted in nominal circumstances just in case they prove handy for theorising about a later anomaly's cause.

Sorry if this comes across as vague,
regards,
Colin Paul Gloster

4. Subject: FlightLinux
Date: Wed, 9 May 2001 14:23:35 -0400
From: David Emery <emery@mitre.org>
To: pstakem@qssmeds.com
CC: kenwood@mitre.org, terry@mitre.org, howell@mitre.org

I'm particularly interested in hearing about issues associated with POSIX (non) Conformance. I've been working with the Open Group on getting the POSIX/Unix and Linux groups to converge. One issue we've had in this is determining exactly where various Linux flavors are not POSIX-conforming.

Also, are you aware of the NSA project to verify a release of Linux? This is very significant in fighting the FUD (Fear/Uncertanty/Doubt) factor associated with Linux as Open Source, by having NSA "prove" a version of Linux is virus-free.

MITRE recently completed an assessment of Open Source for military systems. I presented the attached last week at the Software Technology Conference.

dave

David Emery, The MITRE Corporation

5. Subject: Flight Linux interested
Date: Sun, 19 Aug 2001 17:15:01 +0300
From: George Danchev <danchev@spnet.net>
To: pstakem@qssmeds.com

Hello Dear Sir ,

I'd like to know more about the Flight Linux Project . I've visited the site of it (http://flightlinux.gsfc.nasa.gov) , but the information was not enough for me ... Especially I'm interested in the changes you have made in the Linux kernel , how the embedded Linux kernel is tuned and stuff ...Is FlightLinux a whole distribution , what is changed within the user-space , and can I have it downloaded and installed on my box ?

Thank you in advance ,
Any info about the project will be appreciated

George Danchev,
Air Traffic Controller

6. Subject: rtlinux.org: FlightLinux
Date: Wed, 7 Mar 2001 15:28:13 -0500
From: "Heble, Ajit M" <Ajit.Heble@West.Boeing.com>
To: "'pstakem@qssmeds.com'" <pstakem@qssmeds.com>

Hi Pat!
I am looking into safety critical operating systems and came up with FlightLinux. So what's the current status of it? Could you give me a brief description of its good points/shortcomings? Is it available for download and how much does it cost?

Thanks
Ajit

7. Subject: RE: web page
Date: Fri, 05 Oct 2001 13:06:07 -0400
From: "Kapcio, Paul" <paul.kapcio@baesystems.com>

To: "'Pat Stakem'" <pstakem@qssmeds.com>

What you describe is the new G4 with the new max bus and the new MPC-107. The RAD750 and Power PCI reflect the G3, MPC-106 and 6xx bus.

Motos claims are correct. The max bus (second generation 6xx bus) is what allows them to do multiprocessing.

I'm familiar with the Flightlinux results. We are also working with Lynuxworks to port their Linux product to the RAD750. We've just stated working with them as a result of GSFC's ST-7 contract. Rick Schnuur is the PI for ST-7 and he has asked us to use Linux as the operating system. Since FlightLinux used LynuxWorks, I decided to go with them too. The 560 group we are working with on NGST wants Linux, too.

The limit of 8 processors is the PCI and CompactPCI bus.
Paul

8. Subject: FlightLinux / Europa Orbiter
Date: Fri, 10 Aug 2001 09:41:36 -0700
From: Len Day <len.day@jpl.nasa.gov>
Organization: JPL
To: pstakem@qssmeds.com
CC: Steve Larson <steven.a.larson@jpl.nasa.gov>

Hi Pat,

My name is Len Day at JPL. I am leading a team called Operating System and Avionics Support for Europa Orbiter and someone just emailed me something that had your URL in it.

Our baseline OS is VxWorks but I have been wanting to come up to speed on real-time Linux in the hopes I could demonstrate our software with it on our testbed.

So this is just to get in touch, I'll read your web page to get an idea of where your project is currently. I probably won't have resources to devote to it for a bit but I'd be very interested in working together when the time comes. We currently have several PPC750 testbeds and our flight testbed (also PPC) is supposed to come on line in November.

Len

| Len Day 818-354-4308 |
| Jet Propulsion Lab len.day@jpl.nasa.gov |
----

9. Subject: flight linux

Date: Tue, 20 Mar 2001 13:20:43 -0500
From: marius@umich.edu (Marius Aamodt Eriksen)
To: pstakem@qssmeds.com

hi -

i am with the university of Michigan mars rover project, affiliated with the Mars society. for our rover, we have decided to use a time triggered protocol. this support needs to be integrated in to the linux kernel. i believe that adding support for time triggered protocols in to the linux kernel would be a great achievement, especially for the realtime community.

i am mostly interested in whether the flight linux team would want to join us in developing it, as to ease effort and maximize the expertise put in to building the support in. we currently have one kernel hacker (myself) on the team.

best regards,
marius.

marius@{umich.edu,crockster.net,linux.com}

10. Received: from zeus.edeclaracion.com

Hello!
I have been reading the available information about Flight Linux that i found while i was searching for Linux distributions. I found your project really interesting, and i would to receive some more information about it, i have some questions by the way:

Is it open source? if so is there anyway i can see some source code? I'm interested in contributing in anyway on the project so if is there any change for me please let me know and send me documentation, and all the material concerning to your distribution that i can have access to. thank you for your time.
best regards

--
Miguel A. Bola=F1os
Administrador de Servidores
Vadenu Tecnologias S.A.
Tel: (506) 283-7302
Fax: (506) 283-7421
Email: mike@edeclaracion.com
http://www.edeclaracion.com
--

11. Subject: Re: FlightLinux
Date: Tue, 24 Jul 2001 16:14:42 +0200 (CEST)

From: <pascale@iroe.fi.cnr.it>
To: Pat Stakem <pstakem@qssmeds.com>

Dr. Stakem,

We basically have two platforms. One is based on a i386sx pc-104 by Ampro and the other is a SBC around a pentium 166MMX but we are planning to switch to a mobile pentium (Octagone) since our partners at NASA-nsbf have extensively tested it in flight without reporting problems...

Do you have a time-scale estimation for the release of the software, or at least for a decision about it?

Enzo.

12. Subject: Flight Linux
Date: Sat, 28 Apr 2001 13:21:56 +0100
From: Richard <richard@sheflug.co.uk>
Organization: Sheffield Linux User's Group
To: pstakem@qssmeds.com

Pat

I read on one of the NASA web pages that NASA does welcome requests from people and organizations in other countries for people to do public lectures by NASA personnel.

Do you know of anyone who might be willing to come to the UK and give a talk about the use of Linux in space ?

Thanks,
Richard

Sheffield Linux
User's Group

http://www.sheflug.co.uk

13. Subject: Re: FlightGNU/FlightLinux
Date: Fri, 22 Feb 2002 16:47:16 -0700 (MST)
From: Richard Stallman <rms@gnu.org>
To: pstakem@qssmeds.com

Unfortunately, VxWorks had that priority inversion issue on Mars, and they had to purchase the source code under non-disclosure to resolve it.

Can you point me at an article that tells this story? It sounds like something that could make a good argument to persuade various people to use free software.

One issue we ran into was that satellite software such as ours is considered a "munition" by the State Dept. for export control purposes, and requires extensive review before it can be exported. Unfortunately, we are working with the Brits, who have a handy-dandy spacecraft in orbit that we can use. However, I can't hand a copy of the software to them, legally.

Is that true even if you publish the source code? After all, encryption software used to be called a "munition" too, and I think it still is; but when the source code is published, it can be exported too.

I think you are right in saying we are doing gnu/Linux. We are using a standard Linux distribution, and the gnu tools to do the work.

The companies call them "Linux distributions", but really they are distributions of the GNU/Linux system--GNU, with Linux added. GNU is a Unix-like operating system; Linux is the kernel. Could you possibly call them "GNU/Linux distributions", to help correct this confusion?

Our contribution to this combined system is bigger and older than Linux, but we tend to get forgotten when people call the whole thing "Linux". If you could please call it "GNU/Linux", you would really help us. For instance, if you would write things like this:

a standard GNU/Linux distribution, and the gnu tools to do the work.

Interestingly, the Russian laptops seem to use a Solaris or GNU/Linux, and are much more stable,

There is a vast undercurrent of GNU/Linux usage at NASA, which has lead to

instead of writing just "Linux", that would help us a lot. We really need that help (see http://www.gnu.org/gnu/why-gnu-linux.html).

If you could call the project GNU-Flight or Space GNU/Linux, that would be really nice. We'd like people to know that GNU is in space.

The term "the GNU tools" is also misleading, because it gives people a narrow idea of what the GNU system is. GNU is a lot more than tools, and you are probably using more than tools. GCC, GDB and Make are tools, but Bash, GNU libc, and GNOME are not tools.

In this particular point

One of my favorite free software effort is the ELKS work. Not only did they retrofit Linux to work without memory management, they are still doing development.

it might be correct to say "Linux", because I suspect these changes were in the kernel. Unfortunately, when people use the term "Linux" ambiguously, it is hard to tell what they are really talking about on any given occasion. Distinguishing between Linux and GNU/Linux is a good way to make things clear.

14. Subject: FlightLinux Info
Date: Mon, 04 Jun 2001 14:31:24 +0100
From: Rob Smith <rjsmith@scs.dera.gov.uk>
To: pstakem@qssmeds.com

Dear Sir,

I recently visited your Web site at http://flightlinux.gsfc.nasa.gov/ and am interested in obtaining a copy of the FlightLinux OS.

I am a civil servant working in Space Dept, DERA, UK, currently engaged in software development for CCSDS. The code is being written as a test implementation of a file transfer protocol called CFDP and is designed to run under VxWorks on an ERC-32 architecture. However, the flight experiment on which it will be run, SMX-2 on DERA's STRV-1d satellite, allows new operating systems to be uploaded and hence there is an opportunity to use FlightLinux as an alternative to VxWorks.

I would be grateful if you could let me know whether FlightLinux is currently available and, if so, from where.

I look forward to hearing from you.

Yours faithfully,
Rob Smith

DERA Farnborough,
Farnborough,
Hants,
UK.

15. Subject: Flight Linux discussion
Date: Fri, 10 Aug 2001 09:56:13 -0700
From: Steven A Larson <Steve.Larson@jpl.nasa.gov>
To: pstakem@qssmeds.com

Pat,

Ed Tisdale circulated your URL at JPL. I've looked at it briefly, and it looks like an activity we'd like to know more about.

In his email Ed mentioned that your group was proposing "a roundtable discussion at the Software Engineering Workshop at GSFC in November to explore some of the conflicts and pitfalls between open source and government-sponsored software." Is this discussion going to occur, and if so, what is the scope. Depending on the depth of the discussion (the more detail the better), I think I'd like to send a couple of people out to participate.

Steve

Steve Larson
Europa Orbiter Flight Software Manager
Jet Propulsion Laboratory
Mail Stop 169-315 Voice (818) 354-0679
4800 Oak Grove Dr. Cell (818) 642-8895
Pasadena, CA 91109 FAX (818) 393-4802
----

16. Subject: Re: Flight Linux
Date: Tue, 22 May 2001 08:38:44 -0400
From: "Thomas P. Flatley" <thomas.p.flatley.1@gsfc.nasa.gov>
To: "Thordarson, Sveinn" <Sveinn.Thordarson@trw.com>
CC: pstakem@qssmeds.com

Sveinn,

The person working the "Flight Linux" task is Pat Stakem (pstakem@qssmeds.com). You can contact him directly regarding your interest in trying "Flight Linux" for your application. Let me know if I can do anything to facilitate.

Tom

At 02:25 PM 5/17/01 -0700, you wrote:

Tom -
Per our conversation yesterday, I am e-mailing you to remind you to send the details/contact information regarding Flight Linux.
Thanks.
- Sveinn

Sveinn Thordarson
Electro-Optical Engineer
TRW Space and Electronics Group
One Space Park
Mail Stop R1/1120C

Redondo Beach, CA 90278
(310) 814-6085
Fax: (310) 812-0109
Email: sveinn.thordarson@trw.com <mailto:sveinn.thordarson@trw.com>

17. Subject: FlightLinux: working on a story
Date: Mon, 12 Mar 2001 08:05:54 -0500
From: Tina Gasperson <tinahdee@tampabay.rr.com>
To: pstakem@qssmeds.com

Hello, Pat Stakem,

I'm an editor at Newsforge.com, a site that reports on Open Source news and links to current items of interest to those in the Open Source community.

Do you have time, and would you be interested in sharing with me your thoughts and insights about the FlightLinux project and how it came to be associated with NASA through your proposals; what the status of the project is now; what the future looks like for FlightLinux; your role with it, and your background?

I will use this information for direct quotes and background information for an article I am writing about FlightLinux. If you are able to respond, the article would be slanted mostly toward your involvement with the project.

If you do have the time and inclination, Our readers (and I) would be very appreciative and would be interested in anything you have to say on the subject. If not, we certainly understand.

Thanks, and best wishes,
Tina Gasperson
morning news editor
newsforge.com

Appendix - Development Facilities for the UoSat-12 FlightLinux Flight Demonstration

This section describes the specific development facilities for the UoSat-12 software. These facilities consist of a number of Linux-based Intel boxes located at QSSMEDS for software development, and a breadboard facility for initial code testing. In addition, a breadboard facility acquired from SSTL represents the best model of the actual onboard architecture. This breadboard resides at GSFC and is accessed from our offices at QSSMEDS remotely.

Differences Between Breadboard Units

Parameter	QSS Breadboard	SSTL Breadboard	Flight Unit
Processor	80386sx	80386ex	80386ex
Speed	20, 25, 33,40 Mhz	8, 16, 25 Mhz	8, 16, 25 Mhz
Coprocessor	none	80387sl	80387sl
Main memory	512k-4 megabyte	4 megabyte	4 megabyte
Extended mem	none	32 megabyte	32 megabyte
Async	2	2	not connected
Sync	none	4 full duplex	4 full duplex
Lan	10base-T	10base-T	10base-T
CAN bus	none	yes	yes
Software load	floppy	firmware loader	ViaCAN link, using firmware

The UoSat-12 breadboard and loading facility acquired by and administered by the OMNI Project, GSFC Code 588. This facility consists of a breadboard, flight-like UoSat-12 OBC, a 28-volt bench power supply, a Windows NT-based computer, and associated cables.

NT-Computer Host

This unit is a standard Intel-based desktop PC unit running the Windows-NT operating system and hosting both the "PCAnywhere" software and the SSTL loader program. It is connected to the GSFC LAN and to the OBC breadboard via an asynchronous serial port and a CAN interface with an SSTL-supplied PCI CAN card.

OBC386 Hardware

The OBC is based on Intel's 386EX 32-bit microcontroller. This microcontroller consists of an industry-standard 386SX-microprocessor core and a number of peripherals. The OBC supports a maximum of 4 megabytes of program storage, 128 megabytes of Solid State Data Recorder (referred to as Ramdisk in this document) memory, and 32 kilobytes of firmware storage. The program memory is protected by a majority voting system. Two separate communication controllers handle the serial communication with the up and downlinks. On-board communication is supported by a medium-speed 10BASE-2 Ethernet controller. Telecommand and telemetry is handled by a CAN controller. This controller allows other bus nodes to issue "Reset" commands, switch I/O multiplexers, and enable/disable the Ramdisk without application software support.

386EX Microcontroller

The OBC386 is based on Intel's 386SX-compatible microcontroller. This microcontroller was chosen due to its enhanced microprocessor core, build-in peripherals, and extended addressing capabilities of up to 64 Mbytes. It also provides a software-compatible upgrade path from SSTL's primary OBC186 (based on Intel i80C186). The 386EX is available in commercial and extended temperature range; a military screened 883B version is not available at the time of writing this documentation.

Procedure for Assembling a Test Program

This section describes a procedure to assemble, link, and locate a test program written in Assembly language for the OBC386. The assembler and linker are both Microsoft products whereas the locater is developed by CSI. The batch file assembles the file (command line argument) using the Microsoft assembler**(** MASM) without linking it. The next line links the file using the OBC386 debugger library (OBC386D.LIB). This library contains several routines to read and write to the universal asynchronous receiver-transmitter **(**UART). The OBC386.LIB file (without the "D" extension) implements the same routines but writes to ISCC channel 0 instead. After linking, the file is located to address 0x500 by the CSILOC program. The file can then be uploaded and executed using the WINLOAD utility.

Bench Test Setup

For uploading new software, the UoSat-12 engineering model uses one of the 386EX built-in UART's (TX/RX UART0). This UART is connected to PL21 pin 1 and 2. The second UART is connected to pin 1 and 2 of jumper OPTI2. Both are connected to the host box. The associated Window-NT-based loader software runs under Windows NT.

Software Development Tools

The OBC software is written in two languages: "8086 assembler" and "c." Tools for both are required. For the assembler, two very different variations are found. One is the standard "Intel format" and the other is the AT&T format. These require different assemblers, and use very different syntax. Under the Linux operating system, assemblers and the gnu c compiler (gcc) are readily available.

Remote Operation of the Development Facility

The Windows-NT machine hosting the SSTL loader software is co-located with the breadboard and is interconnected with both asynchronous serial and can busses. The NT box is connected to the GSFC infrastructure LAN and then to the Internet. This equipment is located in the OMNI Laboratory.

Software development is done at QSSMEDS on various desktop Linux machines. A load image is built and stored as a disk file. This is transferred to the NT loader machine by means of the cots "PC-Anywhere" software. This software allows complete console control of the NT system, while maintaining acceptable security. The NT machine may even be rebooted under control of the pc-anywhere. It implements sufficient layers of security and protection to allow remote access over the GSFC LAN. Remote access to the testbed facility provides a level of efficiency, because 1) the development team and the breadboard need not be co-located and 2) the travel requirements are virtually eliminated.